DOCUMENT RESUME

ED 430 541 IR 019 570

AUTHOR Buck, George H.
TITLE Authorware Professional as a Research and Teaching Tool.
PUB DATE 1999-04-22
NOTE 16p.; Paper presented at the Annual Meeting of the American Educational Research Association (Montreal, Quebec, April 22, 1999).
PUB TYPE Reports - Research (143) -- Speeches/Meeting Papers (150)
EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Advanced Courses; *Authoring Aids (Programming); Computer Assisted Instruction; *Computer Science Education; Courseware; Foreign Countries; High Schools; Introductory Courses; Pilot Projects; *Programming; Programming Languages; *Student Attitudes; Teacher Surveys
IDENTIFIERS Alberta; *Authorware

ABSTRACT

A verbal and e-mail survey was conducted of 30 senior high teachers in Alberta who teach computer programming courses in the Career and Technology Studies curriculum, in order to ascertain what courses at the intermediate and advanced levels were being offered, what programming languages were being used, why teachers selected the particular language, the approach used to teach the language, and to gather opinions as to why enrollments drop after the introductory level course. In the fall of 1998, a pilot program was introduced in two senior high schools in Alberta, using Authorware Professional (version 4.0.3) to teach introductory programming. The working hypothesis of the approach was that, by learning beginning programming using Authorware, students would master fundamental logic and procedures quickly by means of the iconic interface. The program of study involved six beginning assignments derived and adapted from a laboratory manual used with fourth-year university undergraduates. Teachers using this approach reported that most students reacted to the program enthusiastically and favorably and that most of the students from the introductory course continued with intermediate-level courses. There are two main obstacles impeding use of Authorware in schools--its cost and the fact that most teachers are ignorant of what it can do. (AEF)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 430 541

TR

Authorware Professional as a research and teaching tool

George H. Buck, Ph.D.

University of Alberta

Presented at the 1999 Annual Meeting

of the

American Educational Research Association

Montréal, Québec

April 22, 1999

IR019570

PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

G.H. Buck

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC).

BEST COPY AVAILABLE

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Authorware Professional as a research and teaching tool

George H. Buck

The authoring language *Authorware*, previously known as *Course of Action*, and latterly also known as *Authorware Professional*, or *Authorware Attain*, was developed by the early 1980s as a means of enabling individuals without interest or proficiency in traditional programming, to create multi-media courseware quickly (Buck, 1992). According to Michael Allen (1996), the designer of Authorware, the language originated from studying how people created instructional material, and from considering what programming background they had. In consequence, an iconic-based interface was developed, that required a minimum of learning by the author before he or she was able to create interactive courseware. Allen wrote, "with only a little knowledge, you can build interactions that will be challenging to program in almost any language"(p. xxix).

In its basic form, Authorware employs a graphical representation of a flowline, onto which are placed icons representing particular types of content and actions, as shown in figure 1.

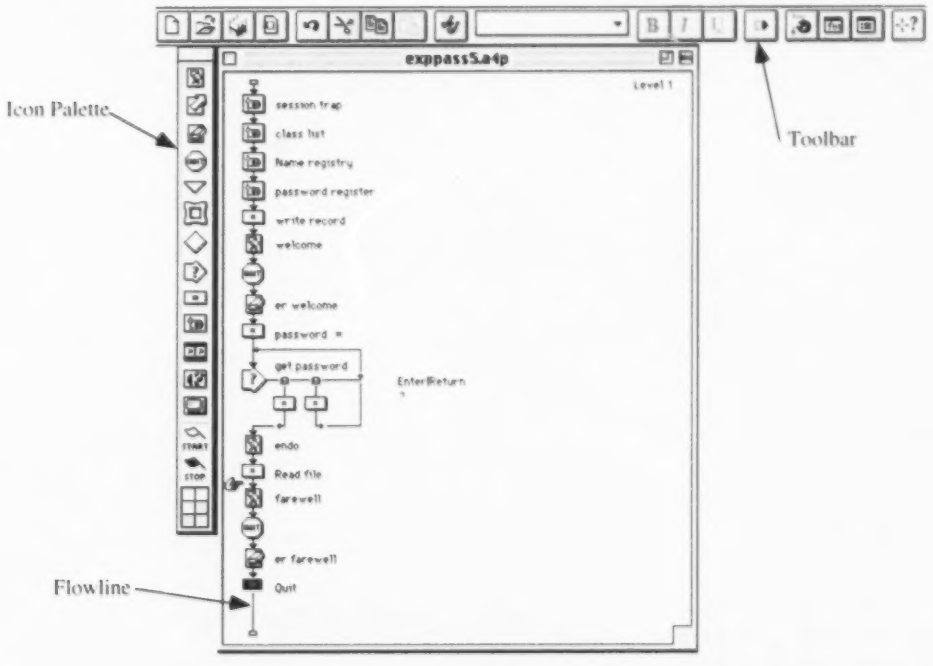

Figure 1. Typical appearance of an Authorware flowline

BEST COPY AVAILABLE

The perception that the flow through the program is entirely linear, as in the manner of programmed instruction or early Skinnerian teaching machines, belies the complex non-linear programming that is created beyond rudimentary stages. The option of flexibility in programming in Authorware means that one may create courseware congruent with theories of learning not usually associate d with computer-assisted instruction. For example, Katz (1996) demonstrated that it was possible to create an instructional program following constructivist tenets. Such flexibility in programming, however, comes at a price. That price is mastering the more arcane programming features in Authorware.

Once the general principles of creating a simple Authorware piece are mastered, one may incorporate more complex features, employing more traditional programming using a high-level language that is built into Authorware. The following figure shows a portion of code entered in a calculation icon to randomize access to an array.

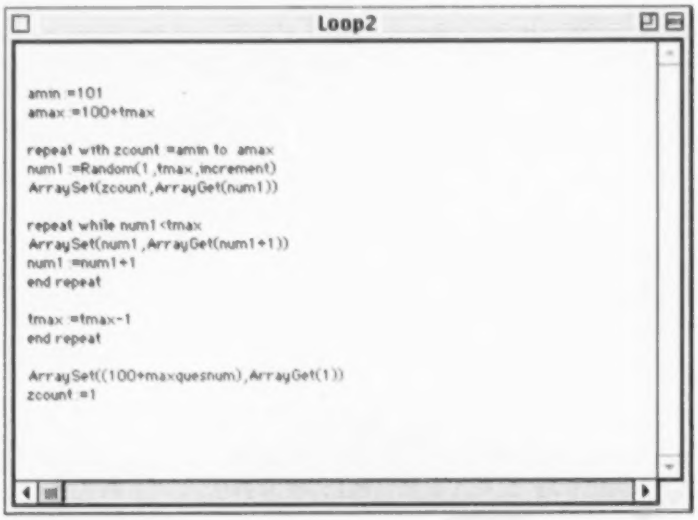

Figure 2. A portion of high-level language code

BEST COPY AVAILABLE

Whether used to create simple interactive presentations, or more elaborate, multi-media courseware, Authorware is generally described as being designed for courseware creation (Alessi & Trollip, 1991; Allen, 1995; Zielinski, 1996).

The Problem

Curricula in many North American schools offer courses or programs that teach aspects of computer programming. While the goals of such courses and programs often differ between jurisdictions, maintaining interest and motivation seem to be universal and perennial concerns (Kleiman, 1982). In Alberta, the responsibility for introducing and instructing computer programming to school age students comes under the purview of the Career and Technology Studies curriculum. In brief, the curriculum is divided into 22 distinct areas or *strands*. Within each strand are a number of courses or modules. Each course entails a minimum 25 hours of instruction and activities by the student. The courses are, in turn, arranged in three levels of difficulty, Introductory, Intermediate, and Advanced. With few exceptions, most computer programming is taught at the senior high school level, as a part of the Information Processing strand. There is no particular language prescribed for use in such courses (Alberta Education, 1998). Language choice is up to the individual school and the teachers in the program. The following scope and sequence chart illustrates the individual computer programming courses available, and their relationship to one another.

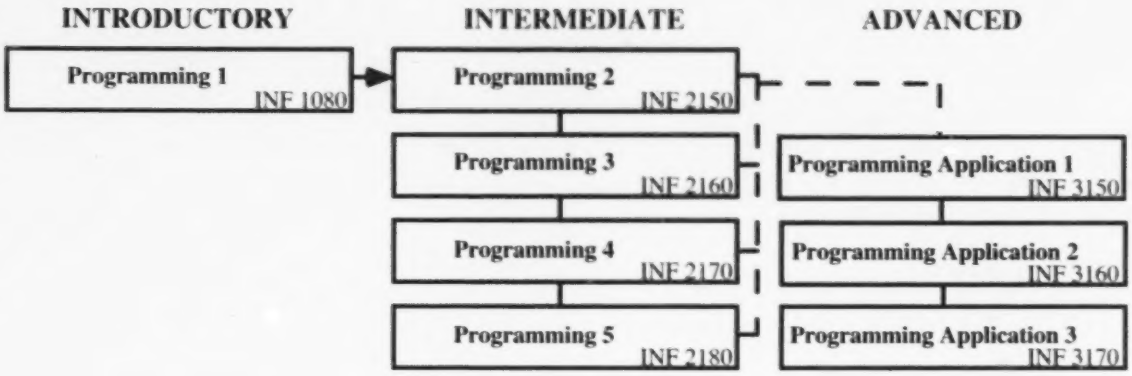

Figure 3. Scope and sequence chart for the Information Processing Strand

Alberta Education (1999), the provincial department of education, compiles enrolment statistics for each course in all the CTS strands. From the time of full implementation of the CTS curriculum in 1997, enrolment in intermediate and advanced level courses in computer programming were consistently less than one half the enrolment in the introductory level course. In most schools, enrolments also decrease after the intermediate level, so much so that there are insufficient numbers to offer advanced level courses. To be sure, some schools report accommodating advanced-level students within offered intermediate-level courses, although this is additional, and often unrecognized work for the teacher. It is important to note that none of the CTS courses are compulsory; all courses are electives.

If, as some computer scientists contend, it is critical for students to gain knowledge of computer programming before entering post-secondary institutions, then adjustments should be made to computer programming courses offered in schools to make them more conducive to retaining students. To address what changes should be made it was first necessary to identify the reasons why students were not electing to take programming courses beyond the introductory level.

Methodology and Data

A verbal and E-mail survey was made of 30 senior high CTS teachers in Alberta , who teach computer programming courses in the CTS program. Of this number, 17 responses were received, yielding a response rate of 56.67%. The purpose of the survey was to ascertain which courses at the intermediate and advanced levels were being offered, what programming language or languages were used, why teachers selected the particular language, the approach used to teach the language, and also to gather opinions from the teachers as to why enrolments drop after the introductory level course.

The teachers who agreed to participate were also asked to gather additional information from students in their introductory classes, using questionnaires and follow-up verbal questioning for clarification. The questions were selected by the author, after initially consulting with the teachers. The questions were: What do you enjoy most about this course?; What do you dislike most about this course?; Why did you select this course?; Do you plan to take a course in this strand at the next level?; What are your reasons for this decision? Three teachers elected not to use questionnaires, instead preferring to pose questions verbally.

Analysis of the data revealed five main factors contributing to students' decisions not to continue with computer programming beyond the introductory level: 1) too steep a learning curve, or sustained difficulty in mastering the programming language; 2) maintaining interest in programming; 3) little apparent practical utility of the course content; 4) no time to take such courses because of the need to take core courses for matriculation; 5) lack of parental support for the courses. While the first two factors reflect student attributes solely, the last three may also reflect parental influence. In other words, if parents do not see the course as being

particularly relevant to their child's future, then they are less likely to encourage enrolment in subsequent courses.

In their comments, several teachers suggested that the apparent lack of parental support resulted from difficulties in demonstrating to parents what had been learned. Even when students create programs, they are often unable to take a copy home and show their parents on their home computer. The reasons for this were either because the required software was not available there, or because the home computer used a different operating system than the computer at school, and the program would not run on more than one platform. Unless the parent visits the school, then, they may possibly see only printouts of the code entered by their child, and this information usually conveys little to the non-programmer about what the program actually does when it is run.

All teachers reported using high-level programming languages, their choice being determined primarily by personal knowledge and experience. The languages used were BASIC, VisualBASIC, COBOL, C++, or in two cases, LOGO. Four teachers, who knew the author and his use of Authorware to develop computer-assisted instruction, reported that their schools possessed Authorware, but that it was not used because they did not know how to use it.

The disparity of languages used is not surprising, given that the requirements of the INF 1080 Introductory course are general, and do not stipulate a particular language, "Students are introduced to computer programming languages and a structured programming environment, and they construct algorithms and code instructions to solve identified problems" (Alberta Education, 1998). Although all of the languages used were high-level, using English terms, the initial degree of difficulty for novices varied considerably, as programming in LOGO, for instance, is generally considered to be easier than mastering the intricacies of COBOL.

In spite of the language selected, the approach to instruction was essentially the same amongst teachers. After being taught essential terms, that at first exposure often seem to be arcane, and an outline of the logic, students would then enter many lines of code following a prescribed procedure in a workbook. The result, assuming that essentials were learned and instructions were followed, was that the student ended up with a program that did something like adding two numbers together that were entered by the user, or that drew a particular object on the screen. Most teachers reported that variations on such themes tend not to hold student attention for very long, so most students' experience with programming is often one of boredom and tedium. This was also a common opinion stated by students. Although some students suggested that they be permitted to select the topic to be addressed in programming, and what features should be included, teachers commented that such an approach would be unsuccessful with most students. The reason being that until students possessed sufficient knowledge and experience in programming, they would encounter problems that the teachers could not solve, and before the teacher could arrive to solve them, the students would lose interest through frustration.

Treatment

In the fall of 1998, a pilot program was introduced in two senior high schools in Alberta, where Authorware Professional, version 4.0.3, would be used to teach introductory programming. These two schools were selected because both possessed site licenses for the full, non-commercial version of Authorware, both had computer labs that could run Authorware, and also because teachers at those schools were sufficiently skilled with Authorware, so that they could instruct without being one step ahead of the students' working knowledge.

The working hypothesis of the approach is that by learning beginning programming by using Authorware, students would master fundamental logic and procedures quickly by means of the iconic interface. Support for this idea comes in part from Bruner's (1964) hierarchy of learning, which is actually a modern iteration of an ancient view of learning (Buck, 1992). Essentially, Bruner (1964) contends that for information to be of interest and use, the information must be encoded in memory in such a way that it is meaningful to the individual. Bruner postulates that the method of encoding follows a hierarchy comprised of three main steps, *enactive, iconic, symbolic*. If an individual cannot relate the symbolic information either to enactive or iconic information already in memory, then it is likely that the symbolic information will be meaningless. This observation helps explain why many students found programming to be boring and disconnected from their reality. By proceeding from an iconic basis then to symbolic representations, in the manner of Bruner, it was anticipated that learning would be facilitated, and interest in the programming would be sustained.

Additionally, the packaging feature and cross-platform design of Authorware, enables students to compile their work, and run the resulting file on almost any home computer, thus showing parents exactly what was being done in programming class. While versions of C can also compile code, the compilations are platform specific. Version 4 of Authorware, as well as the newer version 5, are also capable of altering packaged files so that they may be run across the Internet. No student used this feature with their files because of several factors, lack of server space for student home pages, security concerns of the school boards, and potential slowness of the program run in this way.

The program of study was developed during the summer of 1998. In large part, the six beginning assignments were derived from a laboratory manual used with fourth-year university

undergraduates to teach them the basics of programming in Authorware (Buck &Hunka, 1998). The subject matter and level of complexity was adjusted for beginning high school students. The assignments involve creating several mini tutorials and quizzes. Initial programming consists of the student learning the significance of icons by dragging them to the flowline, running the program and adding content information such as text and graphics. The interface for creating text and graphics resembles common word processors and graphics packages, so knowledge from such software could be transferred from those programs with relative ease. Subsequent assignments entailed the use of icons that present multi-media information, and others that perform animations and motions on the screen.

The concept of variables is introduced in the second assignment. By means of an interaction icon, students asked users for a name. Using a calculation icon, the means for entering high-level code, a simple statement was entered that transfers the text entered by the user to a system variable built into Authorware. The data was subsequently displayed on screen by means of embedding the system variable in a display. Figure 4 shows the general layout and code used to capture a user's name, and how the resulting data is displayed on the screen.

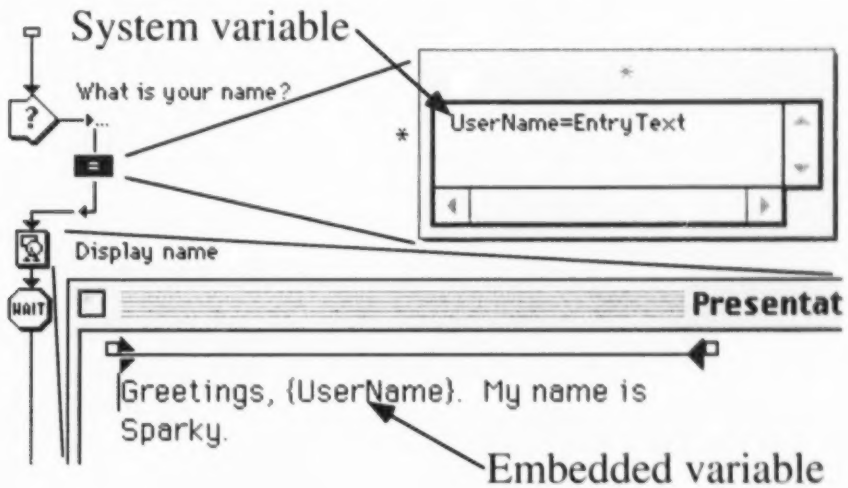

Figure 4. Creation and use of a system variable

BEST COPY AVAILABLE

Next, other built-in variables were introduced, and then the use of custom-made variables of different varieties, character, numeric and logic. Additionally, functions (commands within Authorware) were used to acquaint the students with, among other things, jumping from one part of the program to another in a non-linear fashion (GoTo function), creating arrays, and causing the program to quit at a specified point. Simple sub-routines were also created on the flowline, and accessed by means of functions.

The final exercise in the introductory course, is to duplicate an earlier assignment that calls for the depiction of a circle, followed by lines that create segments in response to user input. Unlike the first exercise, though, the last assignment requires the student to create the circle and segments algorithmically, without the use of any display icons. This exercise, more than any of the others, shows the student clearly the cognitive shift from iconic to symbolic programming. Although the end result of the two assignments is essentially the same, the more rapid speed of the algorithmic program's execution is a tangible manifestation of why most programming languages do not use an iconic interface.

While some of the assignments require students to follow a prescribed sequence and content, most assignments were designed so that students could select whatever topic they desired, thus helping to sustain their interest.

Results and discussion

Although this approach to teaching programming is being used in only two schools at present, and is still in its infancy, teachers using this approach report that most students reacted to the program enthusiastically, and favourably. Additionally, the teachers indicate that most of the students from the Introductory course are either taking, or plan to take Intermediate-level courses.

One of the teachers involved with the study has begun one of the Intermediate-level courses with additional assignments using Authorware before introducing the main programming language, COBOL, thereby easing the transition into a purely symbolic programming language. She also reports that the Intermediate -level course on multi-media authoring, also uses Authorware, but the emphasis in this course is to use the program for its intended purpose rather than as a vehicle for teaching programming logic. Through inquiry and word of mouth, other teachers are showing interest in the approach of using Authorware to introduce students to computer programming.

Interest in Authorware is also encouraged by Alberta Education, which uses the program to create much of the computer-assisted instruction packages for a number of core subjects at both the junior and senior high school levels. Authorware is also used by several commercial training firms in Alberta for creating instructional material, so knowing how to program in Authorware, also provides students with possible employment opportunities.

In spite of such interest in and use of Authorware, there are two main obstacles impeding its use in schools for any purpose. First, is the cost of Authorware itself. Although school districts and even individual schools can negotiate for a site license, the cost is usually much higher than the cost of software that will be used by more teachers, such as word processing and spreadsheet software. Moreover, there are several different versions of Authorware in circulation, and some so-called educational versions restrict programming to fifty icons, and disable several functions. Several teachers responding to the survey indicated that high cost and confusion about which version to purchase discouraged them from considering Authorware at all.

The second obstacle impeding the use of Authorware, is that most teachers are ignorant of what it can do. In part, this condition stems from misconceptions of what it is, and its capabilities. Indeed, several teachers surveyed reported that they thought Authorware was nothing more than a more elaborate version of presentation software such as Macromedia Director, Hyperstudio, or Microsoft PowerPoint. While there are many manuals and tutorials for common software packages such as Microsoft Office, there are comparatively few resources to acquaint the novice with Authorware. Those that exist have tended to be unnecessarily arcane and expansive (see Zielinski, 1996; Roberts, 1997). Recently, Macromedia , the company that presently markets Authorware, has produced a relatively inexpensive book that provides step-by-step lessons, plus a CD-ROM including samples and the restricted educational version of Authorware 4 (Macromedia, 1997). Additional resources, and occasionally some free code, can be found on some web sites, and in the Authorware newsgroup, bit.listserv.authorware. To be sure, almost all of this information is concerned with using Authorware as an authoring tool, rather than as an introductory programming language.

Although Authorware was conceived by educators, there are usually few references to educational applications of the program in schools. To help address this deficiency, a series of summer courses will be offered at the University of Alberta to acquaint teachers with Authorware, as well as other programming languages, and to provide them with accessible and practical resource material. Through teacher specialist councils, it is further anticipated that this method of introducing programming will become more widely known and practiced. The pilot program will continue, as it remains to be seen whether this approach will lead to higher enrolments in the advanced level computer programming courses.

In spite of its cost and limited exposure, Authorware remains a widely used authoring tool, and, therefore, should be found in senior high schools. As this study has shown, if teachers use Authorware to introduce the principles of programming logic, then it seems likely that the fortunes of computer programming courses will continue to improve. It is the students who will reap the biggest gains, however, since they will be introduced to programming in a more pleasant manner than in the past, and this in turn will encourage many to continue with computer studies.

References

Alberta Education. (1998). *Guides for standards and implementation: Information processing*. Edmonton, AB: Author. Available at URL: http://www.gov.ab.ca/educ/cts.

Alberta Education. (1999). Personal communications.

Allen, M.W. (1995). *Authorware models for instructional design*. Englewood Cliffs, NJ: Macromedia and Prentice-Hall.

Allen, M.W. (1996). Forward, in R.S. Zielinski, *Using Macromedia Authorware 3.5*. Indianapolis, IN: Que Corporation.

Alessi, S.M., & Trollip, S.R. (1991). *Computer-based Instruction: Methods and development*. Englewood Cliffs, NJ: Prentice-Hall.

Bruner, J.S. (1964). The course of cognitive growth. *American Psychologist, 19*, 1-15.

Buck, G.H. (1992). *Instructional devices: Development, learning theories, and deployment*. Unpublished Ph.D. dissertation, Department of Educational Psychology, University of Alberta, Edmonton.

Buck, G.H., and Hunka, S.M. (1998). *Introduction to computer-assisted instruction: Educational Psychology 480 laboratory manual*. Edmonton, AB: University of Alberta, Department of Educational Psychology.

Katz, G. (1996). *A constructivist approach to computer-based instruction.*. Unpublished M.Ed. thesis, Department of Educational Psychology, University of Alberta, Edmonton.

Kleiman, G. (1982). Teaching Johnny to program. *Compute!, 4*(7), 88-91.

Macromedia. (1997). *Macromedia Authorware 4 authorized*. Berkeley, CA: Macromedia Press.

Niemeyer, C. (1997). Authorware for computer-assisted instruction. *Library Hi Tech, 15* (1-2), 133-44.

Roberts, N. (1997). *The official guide to Authorware 4*. Berkeley, CA: Peachpit Press.

Zielinski, R.S. (1996). *Using Macromedia Authorware 3.5*. Indianapolis, IN: Que Corporation.